Implementing Trojan-Resilient Hardware from (Mostly) Untrusted Components Designed by Colluding Manufacturers

Olivier Bronchain Louis Dassy
Sebastian Faust François-Xavier Standaert

European Research Council
Established by the European Commission

Outline

Introduction

Private Circuits 3

Targeted algorithm

Hardware Design

Conclusion

Outline

Introduction

Private Circuits 3

Targeted algorithm

Hardware Design

Conclusion

What's the threat ?

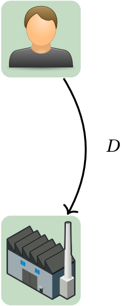
The IC market works as follow:

1. A designer implements specifications D (i.e HDL)

What's the threat ?

The IC market works as follow:

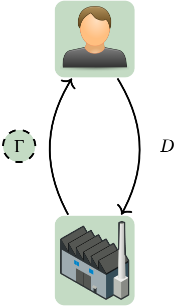
1. A designer implements specifications D (i.e HDL)
2. He sends it to a manufacturers



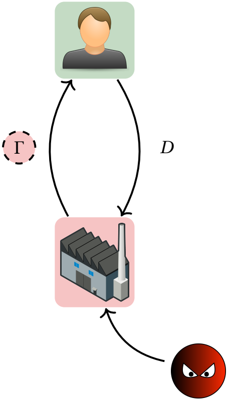
What's the threat ?

The IC market works as follow:

1. A designer implements specifications *D* (i.e HDL)
2. He sends it to a manufacturers
3. He receives the chip Γ corresponding to the specifications *D*.



What's the threat ?



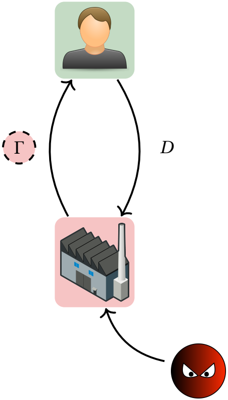
The IC market works as follow:

1. A designer implements specifications *D* (i.e HDL)
2. He sends it to a manufacturers
3. He receives the chip Γ corresponding to the specifications *D*.

What if the manufacturer can not be trusted ?

1. Cheat code: Trojan triggers on predefined inputs

What's the threat ?



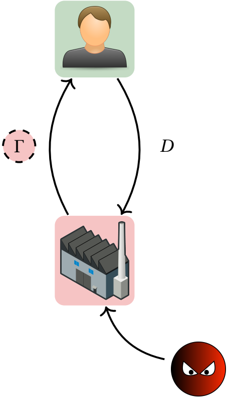
The IC market works as follow:

1. A designer implements specifications D (i.e HDL)
2. He sends it to a manufacturers
3. He receives the chip Γ corresponding to the specifications D.

What if the manufacturer can not be trusted ?

1. Cheat code: Trojan triggers on predefined inputs
2. Time bomb: Trojan triggers after predefined number of executions

What's the threat ?



The IC market works as follow:

1. A designer implements specifications D (i.e HDL)
2. He sends it to a manufacturers
3. He receives the chip Γ corresponding to the specifications D.

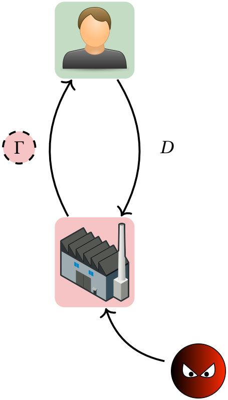
What if the manufacturer can not be trusted ?

1. Cheat code: Trojan triggers on predefined inputs
2. Time bomb: Trojan triggers after predefined number of executions
3. Side-Channel: Trojan triggers on analog signals

What’s the threat ?

The IC market works as follow:

1. A designer implements specifications D (i.e HDL)
2. He sends it to a manufacturers
3. He receives the chip Γ corresponding to the specifications D.



What if the manufacturer can not be trusted ?

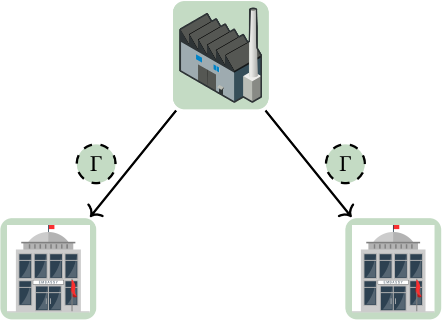
1. Cheat code: Trojan triggers on predefined inputs
2. Time bomb: Trojan triggers after predefined number of executions
3. ~~Side-Channel: Trojan triggers on analog signals~~

Why does it matter ?

1. Two embassies want to build a secure communication channel.

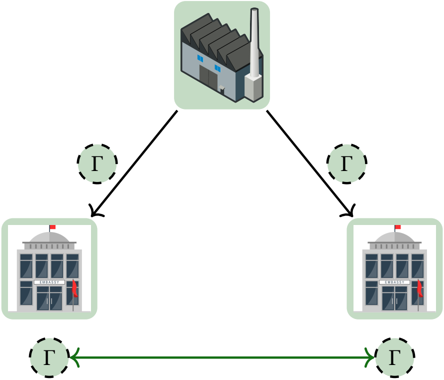
Why does it matter ?

1. Two embassies want to build a secure communication channel.
2. They buy hardware boxes from a foundry.



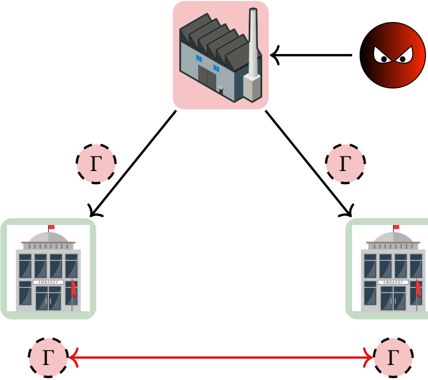
Why does it matter ?

1. Two embassies want to build a secure communication channel.
2. They buy hardware boxes from a foundry.
3. And build the communication channel.



Why does it matter ?

1. Two embassies want to build a secure communication channel.
2. They buy hardware boxes from a foundry.
3. And build the communication channel.
4. The foundry needs to be trusted.



Outline

Introduction

Private Circuits 3

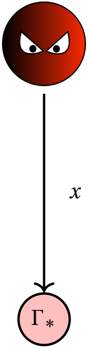
Targeted algorithm

Hardware Design

Conclusion

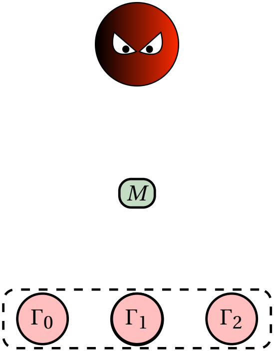
Trojan resilience overview: *Dziembowski et al (CCS2016)*

- Adversary interacting directly with Γ can trigger Trojan.



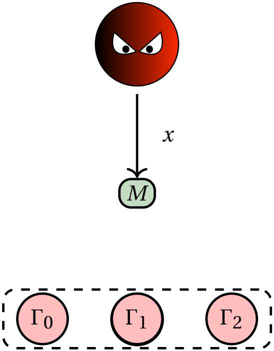
Trojan resilience overview: *Dziembowski et al (CCS2016)*

- Adversary interacting directly with Γ can trigger Trojan.
- He can interact with small trusted circuit M called the master.



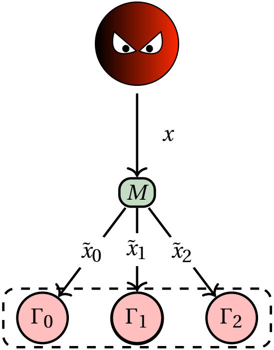
Trojan resilience overview: *Dziembowski et al (CCS2016)*

- Adversary interacting directly with Γ can trigger Trojan.
- He can interact with small trusted circuit M called the master.



Trojan resilience overview: *Dziembowski et al (CCS2016)*

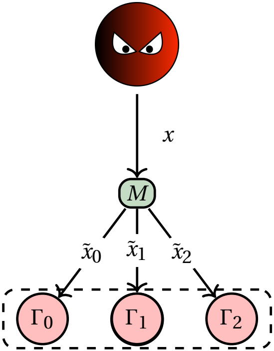
- Adversary interacting directly with Γ can trigger Trojan.
- He can interact with small trusted circuit M called the master.
- **Input scrambling**: randomized inputs to Γ_i that runs three-party computation.

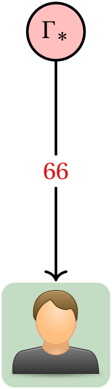


Trojan resilience overview: *Dziembowski et al (CCS2016)*

- Adversary interacting directly with Γ can trigger Trojan.
- He can interact with small trusted circuit M called the master.
- **Input scrambling**: randomized inputs to Γ_i that runs three-party computation.

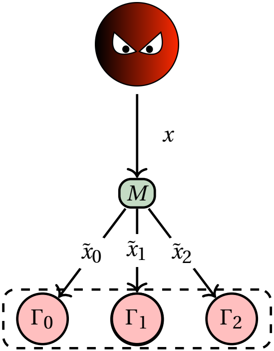
- Single untrusted chip can send malicious output.



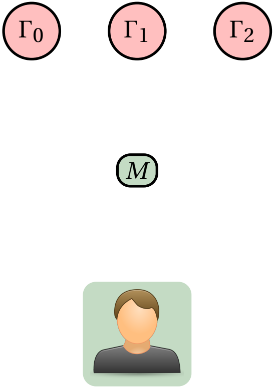


Trojan resilience overview: *Dziembowski et al (CCS2016)*

- Adversary interacting directly with Γ can trigger Trojan.
- He can interact with small trusted circuit M called the master.
- **Input scrambling**: randomized inputs to Γ_i that runs three-party computation.

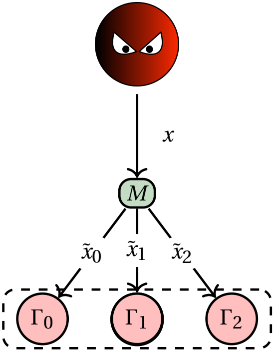


- Single untrusted chip can send malicious output.
- Use **redundancy** in unstrusted devices.

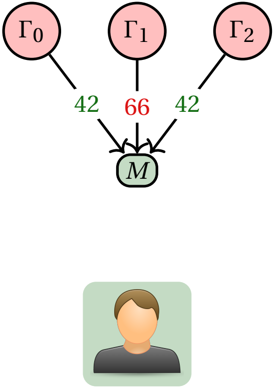


Trojan resilience overview: *Dziembowski et al (CCS2016)*

- Adversary interacting directly with Γ can trigger Trojan.
- He can interact with small trusted circuit M called the master.
- **Input scrambling**: randomized inputs to Γ_i that runs three-party computation.

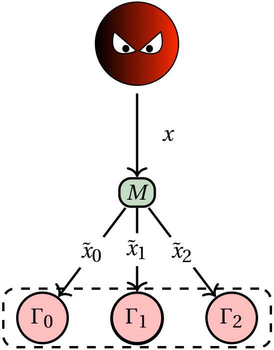


- Single untrusted chip can send malicious output.
- Use **redundancy** in unstrusted devices.

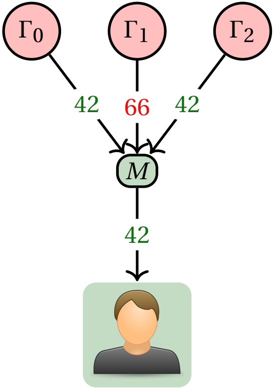


Trojan resilience overview: *Dziembowski et al (CCS2016)*

- Adversary interacting directly with Γ can trigger Trojan.
- He can interact with small trusted circuit M called the master.
- **Input scrambling**: randomized inputs to Γ_i that runs three-party computation.

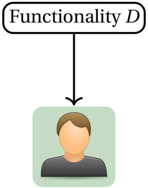


- Single untrusted chip can send malicious output.
- Use **redundancy** in unstrusted devices.
- Perform trusted majority vote among them.



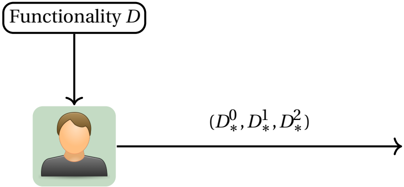
Countermeasure: Compiling and Testing

1. The designer receives some specifications D,



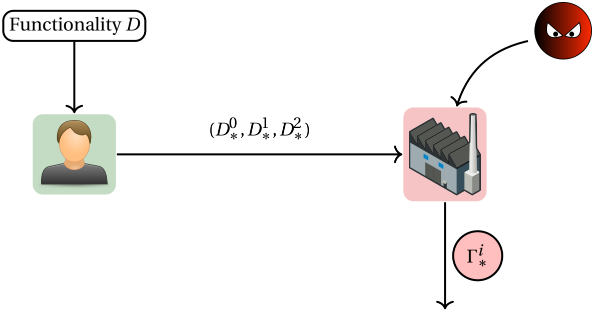
Countermeasure: Compiling and Testing

1. The designer receives some specifications D,
2. Runs a generic compiler and get triplets (D_*^0, D_*^1, D_*^3) called sub-circuits.



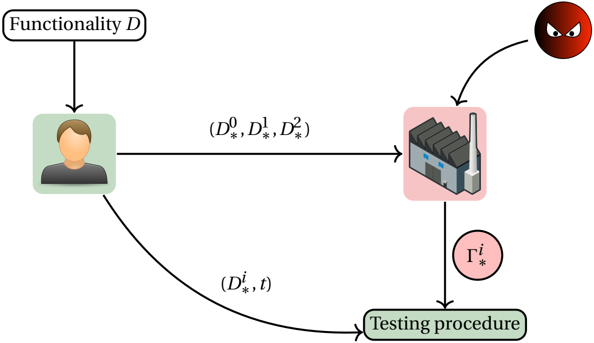
Countermeasure: Compiling and Testing

1. The designer receives some specifications D,
2. Runs a generic compiler and get triplets (D_*^0, D_*^1, D_*^3) called sub-circuits.
3. The untrusted factory builds the circuits and can insert digital Trojan.



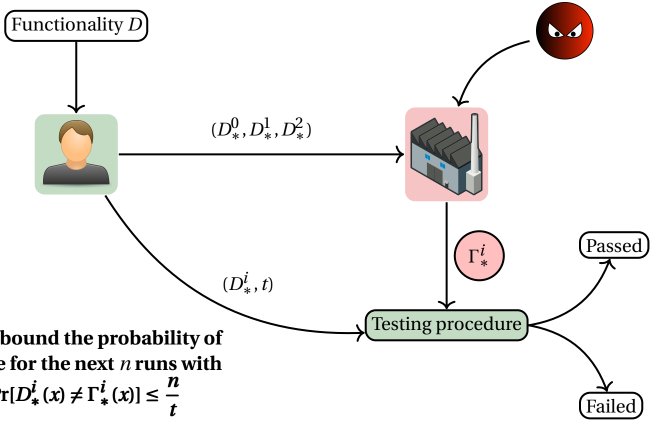
Countermeasure: Compiling and Testing

1. The designer receives some specifications D,
2. Runs a generic compiler and get triplets (D_*^0, D_*^1, D_*^3) called sub-circuits.
3. The untrusted factory builds the circuits and can insert digital Trojan.
4. The circuits are tested a random number of time $t' \leftarrow t$.



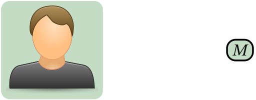
Countermeasure: Compiling and Testing

1. The designer receives some specifications D,
2. Runs a generic compiler and get triplets (D_*^0, D_*^1, D_*^3) called sub-circuits.
3. The untrusted factory builds the circuits and can insert digital Trojan.
4. The circuits are tested a random number of time $t' \leftarrow t$.



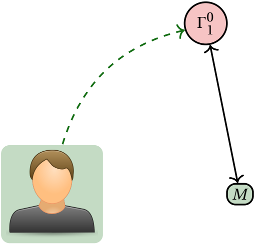
Tests bound the probability of failure for the next n runs with $\Pr[D_*^i(x) \neq \Gamma_*^i(x)] \leq \frac{n}{t}$

Countermeasure second step: Assemble and Run



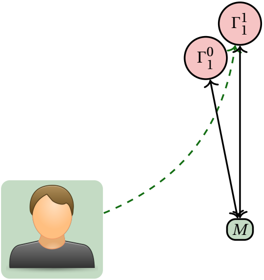
Countermeasure second step: Assemble and Run

- The designer connects the tested circuits,



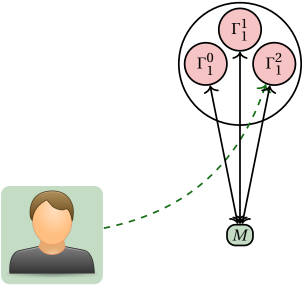
Countermeasure second step: Assemble and Run

- The designer connects the tested circuits,



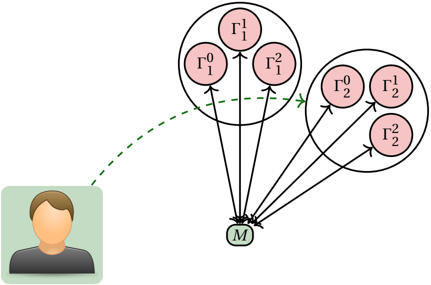
Countermeasure second step: Assemble and Run

- The designer connects the tested circuits,
- He duplicates the triplet λ times.



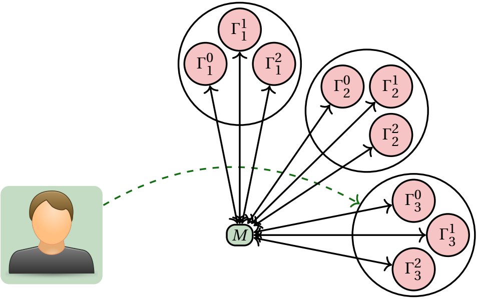
Countermeasure second step: Assemble and Run

- The designer connects the tested circuits,
- He duplicates the triplet λ times.



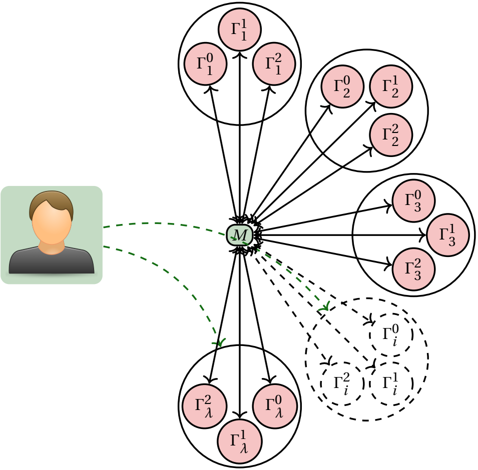
Countermeasure second step: Assemble and Run

- The designer connects the tested circuits,
- He duplicates the triplet λ times.



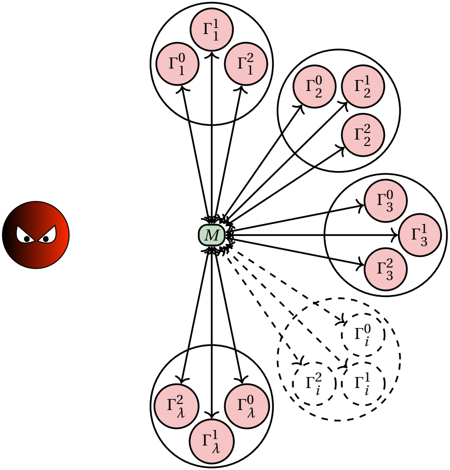
Countermeasure second step: Assemble and Run

- The designer connects the tested circuits,
- He duplicates the triplet λ times.



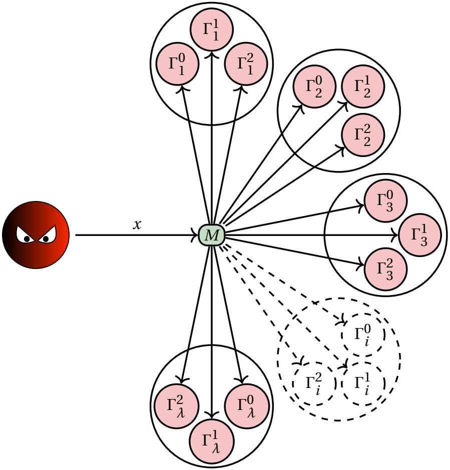
Countermeasure second step: Assemble and Run

- The designer connects the tested circuits,
- He duplicates the triplet λ times.



Countermeasure second step: Assemble and Run

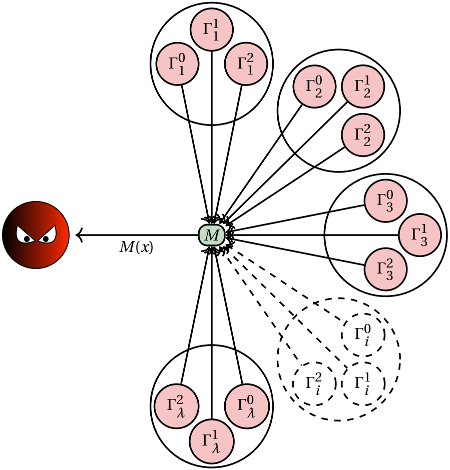
- The designer connects the tested circuits,
- He duplicates the triplet λ times.
- The adversary sends requests to the trusted M that performs secret sharing



Countermeasure second step: Assemble and Run

- The designer connects the tested circuits,
- He duplicates the triplet λ times.
- The adversary sends requests to the trusted M that performs secret sharing
- He receives an incorrect output $M(x)$ with probability:

$$\Pr[M(x) \neq D(x)] = \left(\frac{n}{t}\right)^{\lambda/2}$$

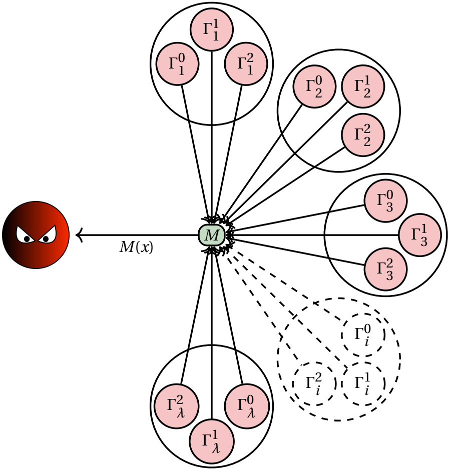


Countermeasure second step: Assemble and Run

- The designer connects the tested circuits,
- He duplicates the triplet λ times.
- The adversary sends requests to the trusted M that performs secret sharing
- He receives an incorrect output $M(x)$ with probability:

$$\Pr[M(x) \neq D(x)] = \left(\frac{n}{t}\right)^{\lambda/2}$$

This property is called **robustness**:



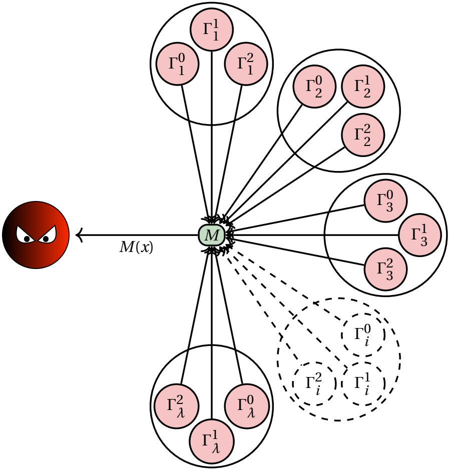
Countermeasure second step: Assemble and Run

- The designer connects the tested circuits,
- He duplicates the triplet λ times.
- The adversary sends requests to the trusted M that performs secret sharing
- He receives an incorrect output $M(x)$ with probability:

$$\Pr[M(x) \neq D(x)] = \left(\frac{n}{t}\right)^{\lambda/2}$$

This property is called **robustness**:

- ☺ Correct value



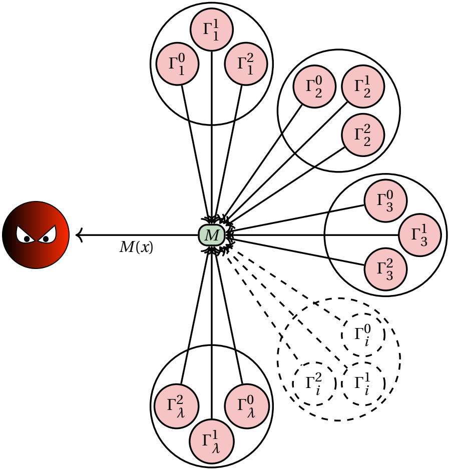
Countermeasure second step: Assemble and Run

- The designer connects the tested circuits,
- He duplicates the triplet λ times.
- The adversary sends requests to the trusted M that performs secret sharing
- He receives an incorrect output $M(x)$ with probability:

$$\Pr[M(x) \neq D(x)] = \left(\frac{n}{t}\right)^{\lambda/2}$$

This property is called **robustness**:

- ☺ Correct value
- ☺ No Denial of Service



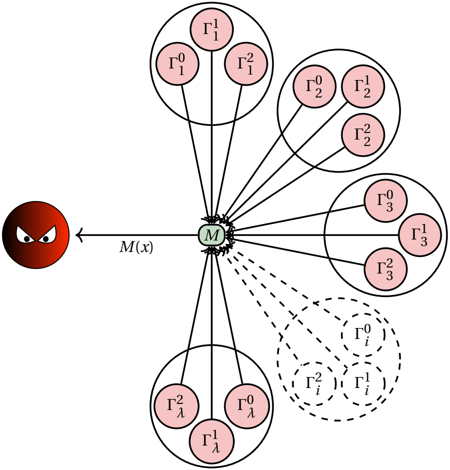
Countermeasure second step: Assemble and Run

- The designer connects the tested circuits,
- He duplicates the triplet λ times.
- The adversary sends requests to the trusted M that performs secret sharing
- He receives an incorrect output $M(x)$ with probability:

$$\Pr[M(x) \neq D(x)] = \left(\frac{n}{t}\right)^{\lambda/2}$$

This property is called **robustness**:

- ☺ Correct value
- ☺ No Denial of Service
- ☹ Limited runs



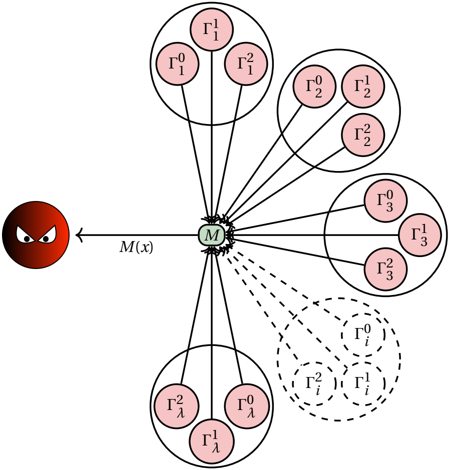
Countermeasure second step: Assemble and Run

- The designer connects the tested circuits,
- He duplicates the triplet λ times.
- The adversary sends requests to the trusted M that performs secret sharing
- He receives an incorrect output $M(x)$ with probability:

$$\Pr[M(x) \neq D(x)] = \left(\frac{n}{t}\right)^{\lambda/2}$$

This property is called **robustness**:

- ☺ Correct value
- ☺ No Denial of Service
- ☹ Limited runs



No need for non colluding manufacturers thanks to combination of:
Passive secure multi-party computation
Test amplification

Design goals

Targeted Robustness level:

$$\Pr[M(x) \neq D(x)] \leq \left(\frac{n}{t}\right)^{\lambda/2} \leq 2^{-80}$$

Design goals

Targeted Robustness level:

$$\Pr[M(x) \neq D(x)] \leq \left(\frac{n}{t}\right)^{\lambda/2} \leq 2^{-80}$$

Methodology:

1. Fix the number of online runs n

Speed

- Get higher encryption throughput

Design goals

Targeted Robustness level:

$$\Pr[M(x) \neq D(x)] \leq \left(\frac{n}{t}\right)^{\lambda/2} \leq 2^{-80}$$

Methodology:

1. Fix the number of online runs n
2. Fix the testing time

Speed

- Get higher encryption throughput
- Increase number of tests t

Design goals

Targeted Robustness level:

$$\Pr[M(x) \neq D(x)] \leq \left(\frac{n}{t}\right)^{\lambda/2} \leq 2^{-80}$$

Methodology:

1. Fix the number of online runs n
2. Fix the testing time
3. Set the redundancy λ to get desired robustness

Speed

- Get higher encryption throughput
- Increase number of tests t

Design goals

Targeted Robustness level:

$$\Pr[M(x) \neq D(x)] \leq \left(\frac{n}{t}\right)^{\lambda/2} \leq 2^{-80}$$

Methodology:

1. Fix the number of online runs n
2. Fix the testing time
3. Set the redundancy λ to get desired robustness

Trusted area

- Reduce the trusted area compared to the unprotected:

AES

Speed

- Get higher encryption throughput
- Increase number of tests t

Design goals

Targeted Robustness level:

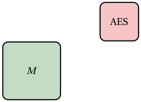
$$\Pr[M(x) \neq D(x)] \leq \left(\frac{n}{t}\right)^{\lambda/2} \leq 2^{-80}$$

Methodology:

1. Fix the number of online runs n
2. Fix the testing time
3. Set the redundancy λ to get desired robustness

Trusted area

- Reduce the trusted area compared to the unprotected:



Speed

- Get higher encryption throughput
- Increase number of tests t

Design goals

Targeted Robustness level:

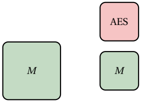
$$\Pr[M(x) \neq D(x)] \leq \left(\frac{n}{t}\right)^{\lambda/2} \leq 2^{-80}$$

Methodology:

1. Fix the number of online runs n
2. Fix the testing time
3. Set the redundancy λ to get desired robustness

Trusted area

- Reduce the trusted area compared to the unprotected:



Speed

- Get higher encryption throughput
- Increase number of tests t

Design goals

Targeted Robustness level:

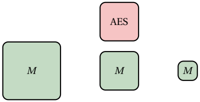
$$\Pr[M(x) \neq D(x)] \leq \left(\frac{n}{t}\right)^{\lambda/2} \leq 2^{-80}$$

Methodology:

1. Fix the number of online runs n
2. Fix the testing time
3. Set the redundancy λ to get desired robustness

Trusted area

- Reduce the trusted area compared to the unprotected:



Speed

- Get higher encryption throughput
- Increase number of tests t

Design goals

Targeted Robustness level:

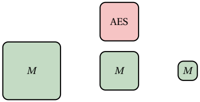
$$\Pr[M(x) \neq D(x)] \leq \left(\frac{n}{t}\right)^{\lambda/2} \leq 2^{-80}$$

Methodology:

1. Fix the number of online runs n
2. Fix the testing time
3. Set the redundancy λ to get desired robustness

Trusted area

- Reduce the trusted area compared to the unprotected:



- Simple computation carried out by M.

Speed

- Get higher encryption throughput
- Increase number of tests t

Design goals

Targeted Robustness level:

$$\Pr[M(x) \neq D(x)] \leq \left(\frac{n}{t}\right)^{\lambda/2} \leq 2^{-80}$$

Methodology:

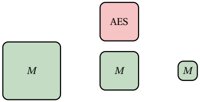
1. Fix the number of online runs n
2. Fix the testing time
3. Set the redundancy λ to get desired robustness

Speed

- Get higher encryption throughput
- Increase number of tests t

Trusted area

- Reduce the trusted area compared to the unprotected:

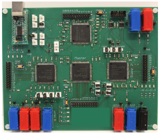


- Simple computation carried out by M.

Contributions

1. First Private Circuit 3 implementation
2. Algorithm: protocol and blockcipher
3. One order of magnitude smaller trusted circuits

Design goals



Targeted Robustness level:

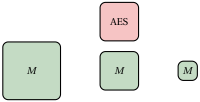
$$\Pr[M(x) \neq D(x)] \leq \left(\frac{n}{t}\right)^{\lambda/2} \leq 2^{-80}$$

Methodology:

1. Fix the number of online runs n
2. Fix the testing time
3. Set the redundancy λ to get desired robustness

Trusted area

- Reduce the trusted area compared to the unprotected:



- Simple computation carried out by M.

Speed

- Get higher encryption throughput
- Increase number of tests t

Contributions

1. First Private Circuit 3 implementation
2. Algorithm: protocol and blockcipher
3. One order of magnitude smaller trusted circuits

Outline

Introduction

Private Circuits 3

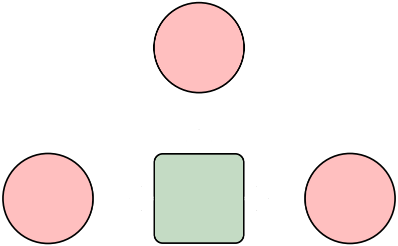
Targeted algorithm

Hardware Design

Conclusion

Generic operations for a single sub-circuit: *Araki et al. (CCS 2016)*

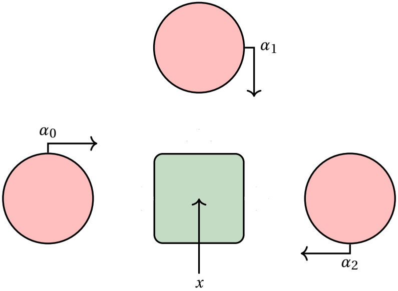
Secret Sharing



Generic operations for a single sub-circuit: *Araki et al. (CCS 2016)*

Secret Sharing

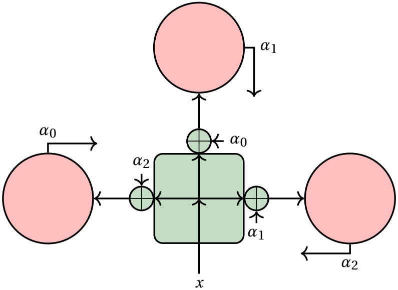
- On the reception of x, circuits send random values α_i.



Generic operations for a single sub-circuit: *Araki et al. (CCS 2016)*

Secret Sharing

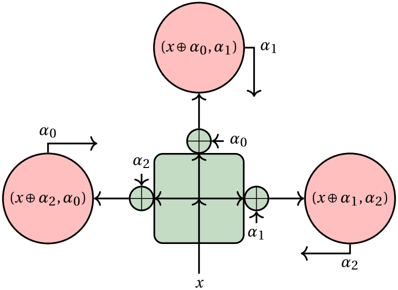
- On the reception of x, circuits send random values α_i.
- The secret is shared.



Generic operations for a single sub-circuit: *Araki et al. (CCS 2016)*

Secret Sharing

- On the reception of x, circuits send random values α_i.
- The secret is shared.

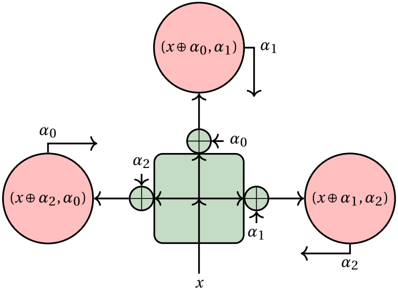


Secret Addition

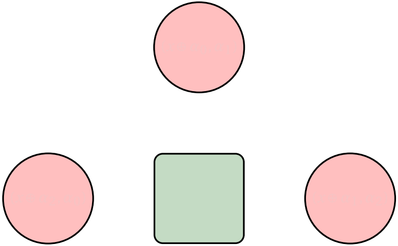
Generic operations for a single sub-circuit: *Araki et al. (CCS 2016)*

Secret Sharing

- On the reception of x, circuits send random values α_i.
- The secret is shared.



Secret Multiplication



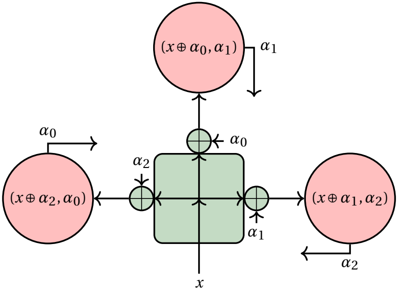
Generic operations for a single sub-circuit: *Araki et al. (CCS 2016)*

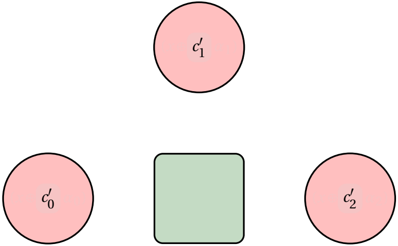
Secret Sharing

- On the reception of x, circuits send random values α_i.
- The secret is shared.

Secret Multiplication

- Circuit computes c'_i based on two shares to multiply.

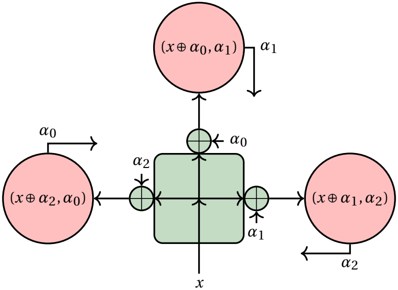




Generic operations for a single sub-circuit: *Araki et al. (CCS 2016)*

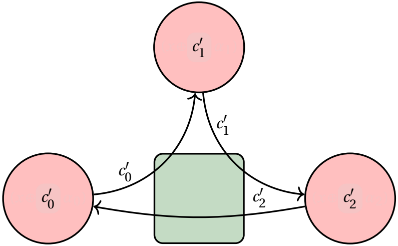
Secret Sharing

- On the reception of x, circuits send random values α_i.
- The secret is shared.



Secret Multiplication

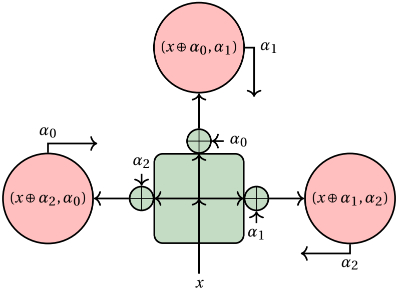
- Circuit computes c'_i based on two shares to multiply.
- This value is sent to the following mini-circuits.



Generic operations for a single sub-circuit: *Araki et al. (CCS 2016)*

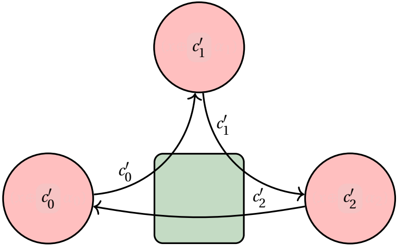
Secret Sharing

- On the reception of x, circuits send random values α_i.
- The secret is shared.



Secret Multiplication

- Circuit computes c_i' based on two shares to multiply.
- This value is sent to the following mini-circuits.



Can build any functions

- No memory is needed in the master, only rooting and xoring
- A single element needs to be sent for multiplication

Blockcipher: Rijndael AES Sbox

Speed depends on:

- Number of rounds
- Number of Sbox's per round
- Bits to exchange per Sbox.

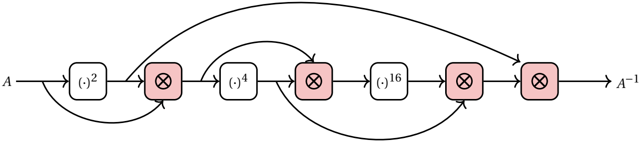
Cipher	# of **rounds**	**Sbox per round**	**bits per enc.**

Blockcipher: Rijndael AES Sbox

Speed depends on:

- Number of rounds
- Number of Sbox's per round
- Bits to exchange per Sbox.

Cipher	# of **rounds**	**Sbox per round**	**bits per enc.**



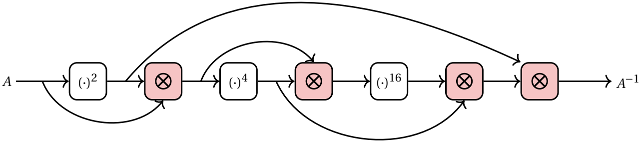
Naive AES Sbox

Blockcipher: Rijndael AES Sbox

Speed depends on:

- Number of rounds
- Number of Sbox's per round
- Bits to exchange per Sbox.

Cipher	# of **rounds**	**Sbox per round**	**bits per enc.**
AES GF(2^8)	10	16	5,120



4 mult. over
GF(2^8)
32-bits transfer

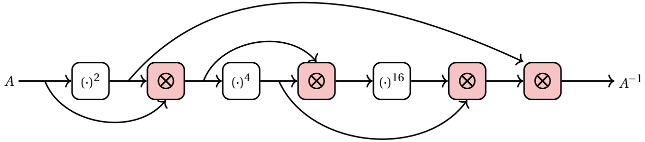
Naive AES Sbox

Blockcipher: Rijndael AES Sbox

Speed depends on:

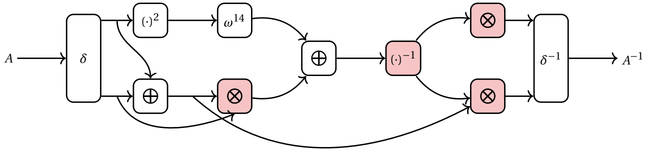
- Number of rounds
- Number of Sbox's per round
- Bits to exchange per Sbox.

Cipher	# of **rounds**	**Sbox per round**	**bits per enc.**
AES GF(2^8)	10	16	5,120



4 mult. over GF(2^8)
32-bits transfer

Naive AES Sbox



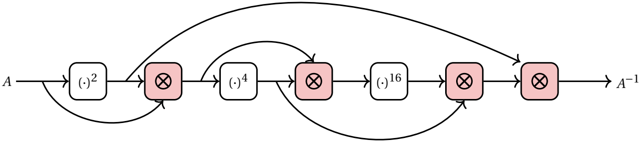
Advanced AES Sbox

Blockcipher: Rijndael AES Sbox

Speed depends on:

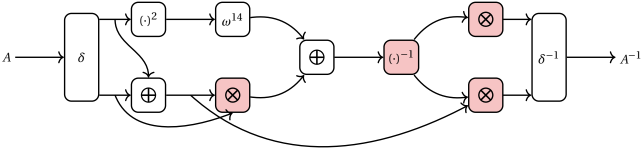
- Number of rounds
- Number of Sbox's per round
- Bits to exchange per Sbox.

Cipher	# of **rounds**	**Sbox per round**	**bits per enc.**
AES GF(2^8)	10	16	5,120
AES GF($(2^4)^2$)	10	16	3,200



4 mult. over
GF(2^8)
32-bits transfer

Naive AES Sbox



5 mult. over
GF(2^4)
20-bits transfer

Advanced AES Sbox

Blockcipher: Mysterion Sbox

Speed depends on:

- Number of rounds
- Number of Sbox's per round
- Bits to exchange per Sbox.

Cipher	# of **rounds**	**Sbox per round**	**bits per enc.**
AES GF(2^8)	10	16	5,120
AES GF($(2^4)^2$)	10	16	3,200

Optimized blockcipher Mysterion:

Blockcipher: Mysterion Sbox

Speed depends on:

- Number of rounds
- Number of Sbox's per round
- Bits to exchange per Sbox.

Cipher	# of **rounds**	**Sbox per round**	**bits per enc.**
AES GF(2^8)	10	16	5,120
AES GF($(2^4)^2$)	10	16	3,200

Optimized blockcipher Mysterion:

- Mysterion is 128 bits blockcipher based on SPN

Blockcipher: Mysterion Sbox

Speed depends on:

- Number of rounds
- Number of Sbox's per round
- Bits to exchange per Sbox.

Cipher	# of **rounds**	**Sbox per round**	**bits per enc.**
AES GF(2^8)	10	16	5,120
AES GF($(2^4)^2$)	10	16	3,200

Optimized blockcipher Mysterion:

- Mysterion is 128 bits blockcipher based on SPN
- Designed for efficient bitslice masking

Blockcipher: Mysterion Sbox

Speed depends on:

- Number of rounds
- Number of Sbox's per round
- Bits to exchange per Sbox.

Cipher	# of **rounds**	**Sbox per round**	**bits per enc.**
AES GF(2^8)	10	16	5,120
AES GF($(2^4)^2$)	10	16	3,200

Optimized blockcipher Mysterion:

- Mysterion is 128 bits blockcipher based on SPN
- Designed for efficient bitslice masking
- With low number of multiplications

Blockcipher: Mysterion Sbox

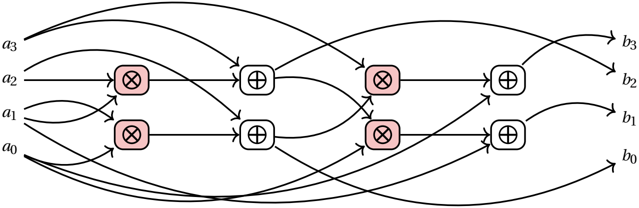
Speed depends on:

- Number of rounds
- Number of Sbox's per round
- Bits to exchange per Sbox.

Cipher	# of **rounds**	**Sbox per round**	**bits per enc.**
AES GF(2^8)	10	16	5,120
AES GF($(2^4)^2$)	10	16	3,200

Optimized blockcipher Mysterion:

- Mysterion is 128 bits blockcipher based on SPN
- Designed for efficient bitslice masking
- With low number of multiplications



Mysterion Sbox

Blockcipher: Mysterion Sbox

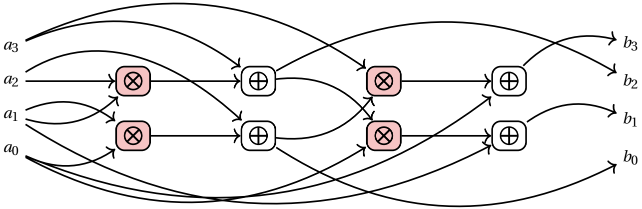
Speed depends on:

- Number of rounds
- Number of Sbox's per round
- Bits to exchange per Sbox.

Cipher	# of **rounds**	**Sbox per round**	**bits per enc.**
AES GF(2^8)	10	16	5,120
AES GF($(2^4)^2$)	10	16	3,200
Mysterion	12	32	1,536

Optimized blockcipher Mysterion:

- Mysterion is 128 bits blockcipher based on SPN
- Designed for efficient bitslice masking
- With low number of multiplications



4 mult. over GF(2)
4-bits transfer

Mysterion Sbox

Blockcipher: Mysterion Sbox

Speed depends on:

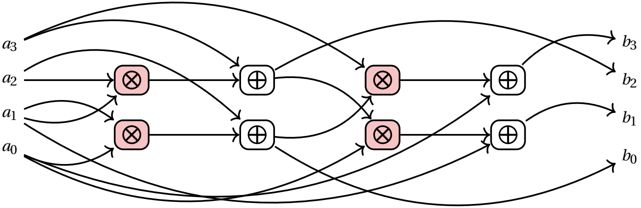
- Number of rounds
- Number of Sbox's per round
- Bits to exchange per Sbox.

Cipher	# of **rounds**	**Sbox per round**	**bits per enc.**
AES GF(2^8)	10	16	5,120
AES GF($(2^4)^2$)	10	16	3,200
Mysterion	12	32	1,536

Optimized blockcipher Mysterion:

- Mysterion is 128 bits blockcipher based on SPN
- Designed for efficient bitslice masking
- With low number of multiplications

Bottleneck if fed fast enough
No restriction on the mini-circuits



4 mult. over GF(2)
4-bits transfer

Mysterion Sbox

Outline

Introduction

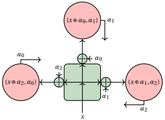
Private Circuits 3

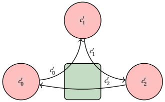
Targeted algorithm

Hardware Design

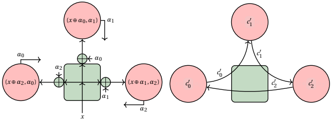
Conclusion

What kind of bus ?



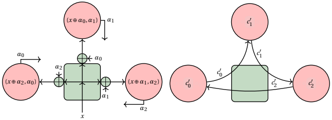


What kind of bus ?



The bus needs to be full duplex to avoid registers in the trusted circuit.

What kind of bus ?

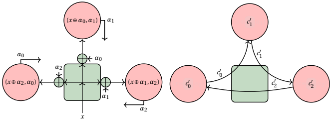


The bus needs to be full duplex to avoid registers in the trusted circuit.

Parallel bus on N bits

Serial bus on 1 bits

What kind of bus ?



The bus needs to be full duplex to avoid registers in the trusted circuit.

Parallel bus on *N* bits

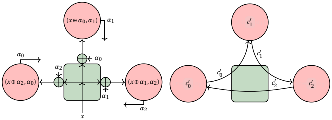
- Allows to send *N* bits at the time in a full duplex manner .

Serial bus on 1 bits

- Allows to send 1 bits at the time in a full duplex manner.

Bus		**AES**		**Mysterion**	
Throug. [Gbps]	**N** [bit]	**Cycles** [cycle]	**Through.** [Mbps]	**Cycles** [cycle]	**Through.** [Mbps]
1.5	1	180	55	96	107
6	4	46	222	24	428

What kind of bus ?



The bus needs to be full duplex to avoid registers in the trusted circuit.

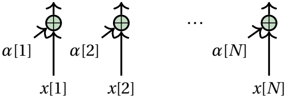
Parallel bus on N bits

- Allows to send N bits at the time in a full duplex manner .
- The master needs to duplicate N times XOR gates and routing.

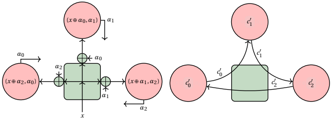
Serial bus on 1 bits

- Allows to send 1 bits at the time in a full duplex manner.
- The master do not need to duplicate XOR gates and routing ✓ .

Bus		**AES**		**Mysterion**	
Throug. [Gbps]	**N** [bit]	**Cycles** [cycle]	**Through.** [Mbps]	**Cycles** [cycle]	**Through.** [Mbps]
1.5	1	180	55	96	107
6	4	46	222	24	428



What kind of bus ?



The bus needs to be full duplex to avoid registers in the trusted circuit.

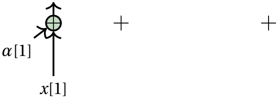
Parallel bus on *N* bits

- Allows to send *N* bits at the time in a full duplex manner .
- The master needs to duplicate *N* times XOR gates and routing.

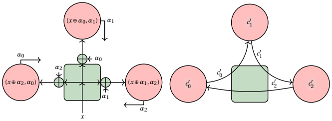
Serial bus on 1 bits

- Allows to send 1 bits at the time in a full duplex manner.
- The master do not need to duplicate XOR gates and routing ✓ .

Bus		**AES**		**Mysterion**	
Throug. [Gbps]	**N** [bit]	**Cycles** [cycle]	**Through.** [Mbps]	**Cycles** [cycle]	**Through.** [Mbps]
1.5	1	180	55	96	107
6	4	46	222	24	428



What kind of bus ?



The bus needs to be full duplex to avoid registers in the trusted circuit.

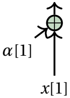
Parallel bus on N bits

- Allows to send N bits at the time in a full duplex manner .
- The master needs to duplicate N times XOR gates and routing.

Serial bus on 1 bits

- Allows to send 1 bits at the time in a full duplex manner.
- The master do not need to duplicate XOR gates and routing ✓.

Bus		**AES**		**Mysterion**	
Throug. [Gbps]	**N** [bit]	**Cycles** [cycle]	**Through.** [Mbps]	**Cycles** [cycle]	**Through.** [Mbps]
1.5	1	180	55	96	107
6	4	46	222	24	428



The bus used is Serial to have 16[GE] per sub-circuit ✓

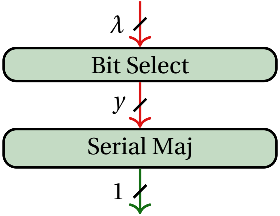
The number of bits to exchange is the critical part for data throughput.

Majority vote

The Majority vote among λ bits
is the most expensive operation.

Majority vote

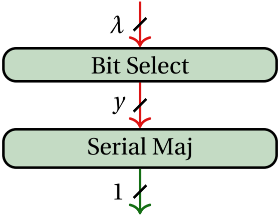
The Majority vote among λ bits
is the most expensive operation.



Majority vote

The Majority vote among λ bits is the most expensive operation.

$$\lambda = 1 \xrightarrow{\text{Enc.}} \xrightarrow{\text{Maj.}}$$



Extrapolated AES data throughput

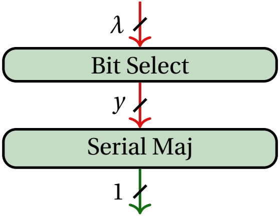
λ	**1**
Throug. [Mbps]	55

Majority vote

The Majority vote among λ bits is the most expensive operation.

$$\lambda = 1 \quad \xrightarrow{\text{Enc.}} \; \xrightarrow{\text{Maj.}}$$

$$\lambda = 4 \quad \xrightarrow{\text{Enc.}} \; \xrightarrow{\text{Maj.}}$$

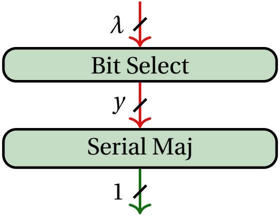


Extrapolated AES data throughput

λ	**1**	**2**	**4**
Throug. [Mbps]	55	51.4	46.3

Majority vote

The Majority vote among λ bits is the most expensive operation.



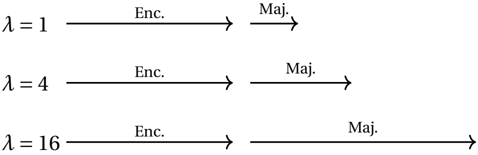
λ

Bit Select

y

Serial Maj

1



$\lambda = 1$ Enc. Maj.

$\lambda = 4$ Enc. Maj.

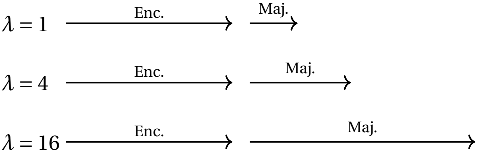
$\lambda = 16$ Enc. Maj.

Extrapolated AES data throughput

$\boldsymbol{\lambda}$	**1**	**2**	**4**	**8**	**16**
Throug. [Mbps]	55	51.4	46.3	38.7	29.1

Majority vote

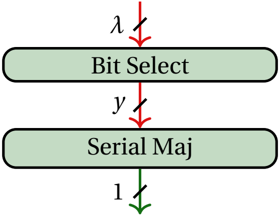
The Majority vote among λ bits is the most expensive operation.



$\lambda = 1$ Enc. Maj.

$\lambda = 4$ Enc. Maj.

$\lambda = 16$ Enc. Maj.



λ

Bit Select

y

Serial Maj

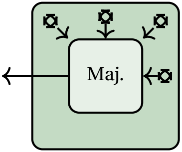
1

Extrapolated AES data throughput

$\boldsymbol{\lambda}$	**1**	**2**	**4**	**8**	**16**
Throug. [Mbps]	55	51.4	46.3	38.7	29.1

Majority vote area [GEs]

λ	**Bit select.**	**Serial Maj.**	**Total**
8	44	52	96
16	77.6	67	144.6



Maj.

Robustness bounds

Methodology:

$$\Pr[M(x) \neq D(x)] = \left(\frac{n}{t}\right)^{\lambda/2} \leq 2^{-80}$$

Robustness bounds

t [days]	**n** [bits]	**AES**			**Mysterion**		
		λ	**ROB.**	**Area** [GEs]	λ	**ROB.**	**Area** [GEs]
1	10^3	6	2^{-95}	181	5	2^{-81}	144
	10^6	8	2^{-86}	224	8	2^{-89}	224
	10^9	15	2^{-85}	380	14	2^{-84}	361
7	10^3	5	2^{-86}	144	5	2^{-88}	144
	10^6	7	2^{-86}	202	7	2^{-88}	202
	10^9	12	2^{-85}	321	11	2^{-81}	286

Robustness bounds

Methodology:

- Set the testing time *t*

$$\Pr[M(x) \neq D(x)] = \left(\frac{n}{t}\right)^{\lambda/2} \leq 2^{-80}$$

Robustness bounds

t [days]	**n** [bits]	**AES**			**Mysterion**		
		λ	**ROB.**	**Area** [GEs]	λ	**ROB.**	**Area** [GEs]
1	10^3	6	2^{-95}	181	5	2^{-81}	144
	10^6	8	2^{-86}	224	8	2^{-89}	224
	10^9	15	2^{-85}	380	14	2^{-84}	361
7	10^3	5	2^{-86}	144	5	2^{-88}	144
	10^6	7	2^{-86}	202	7	2^{-88}	202
	10^9	12	2^{-85}	321	11	2^{-81}	286

Robustness bounds

Methodology:

- Set the testing time t
- Set the number of runs n

$$\Pr[M(x) \neq D(x)] = \left(\frac{n}{t}\right)^{\lambda/2} \leq 2^{-80}$$

Robustness bounds

t [days]	**n** [bits]	**AES**			**Mysterion**		
		λ	**ROB.**	**Area** [GEs]	λ	**ROB.**	**Area** [GEs]
1	10^3	6	2^{-95}	181	5	2^{-81}	144
	10^6	8	2^{-86}	224	8	2^{-89}	224
	10^9	15	2^{-85}	380	14	2^{-84}	361
7	10^3	5	2^{-86}	144	5	2^{-88}	144
	10^6	7	2^{-86}	202	7	2^{-88}	202
	10^9	12	2^{-85}	321	11	2^{-81}	286

Robustness bounds

Methodology:

- Set the testing time t
- Set the number of runs n
- Choose λ

$$\Pr[M(x) \neq D(x)] = \left(\frac{n}{t}\right)^{\lambda/2} \leq 2^{-80}$$

Robustness bounds

t [days]	**n** [bits]	**AES**			**Mysterion**		
		λ	**ROB.**	**Area** [GEs]	λ	**ROB.**	**Area** [GEs]
1	10^3	6	2^{-95}	181	5	2^{-81}	144
	10^6	8	2^{-86}	224	8	2^{-89}	224
	10^9	15	2^{-85}	380	14	2^{-84}	361
7	10^3	5	2^{-86}	144	5	2^{-88}	144
	10^6	7	2^{-86}	202	7	2^{-88}	202
	10^9	12	2^{-85}	321	11	2^{-81}	286

Robustness bounds

Methodology:

- Set the testing time t
- Set the number of runs n
- Choose λ

$$\Pr[M(x) \neq D(x)] = \left(\frac{n}{t}\right)^{\lambda/2} \leq 2^{-80}$$

Changing the blockcipher allows to either:

- Reduce λ

Robustness bounds

t [days]	**n** [bits]	**AES**			**Mysterion**		
		λ	**ROB.**	**Area** [GEs]	λ	**ROB.**	**Area** [GEs]
1	10^3	6	2^{-95}	181	5	2^{-81}	144
	10^6	8	2^{-86}	224	8	2^{-89}	224
	10^9	15	2^{-85}	380	14	2^{-84}	361
7	10^3	5	2^{-86}	144	5	2^{-88}	144
	10^6	7	2^{-86}	202	7	2^{-88}	202
	10^9	12	2^{-85}	321	11	2^{-81}	286

Robustness bounds

Methodology:

- Set the testing time t
- Set the number of runs n
- Choose λ

$$\Pr[M(x) \neq D(x)] = \left(\frac{n}{t}\right)^{\lambda/2} \leq 2^{-80}$$

Changing the blockcipher allows to either:

- Reduce λ

Robustness bounds

t [days]	**n** [bits]	**AES**			**Mysterion**		
		λ	**ROB.**	**Area** [GEs]	λ	**ROB.**	**Area** [GEs]
1	10^3	6	2^{-95}	181	5	2^{-81}	144
	10^6	8	2^{-86}	224	8	2^{-89}	224
	10^9	15	2^{-85}	380	14	2^{-84}	361
7	10^3	5	2^{-86}	144	5	2^{-88}	144
	10^6	7	2^{-86}	202	7	2^{-88}	202
	10^9	12	2^{-85}	321	11	2^{-81}	286

Robustness bounds

Methodology:

- Set the testing time t
- Set the number of runs n
- Choose λ

$$\Pr[M(x) \neq D(x)] = \left(\frac{n}{t}\right)^{\lambda/2} \leq 2^{-80}$$

Changing the blockcipher allows to either:

- Reduce λ

Robustness bounds

t [days]	**n** [bits]	**AES**			**Mysterion**		
		λ	**ROB.**	**Area** [GEs]	λ	**ROB.**	**Area** [GEs]
1	10^3	6	2^{-95}	181	5	2^{-81}	144
	10^6	8	2^{-86}	224	8	2^{-89}	224
	10^9	15	2^{-85}	380	14	2^{-84}	361
7	10^3	5	2^{-86}	144	5	2^{-88}	144
	10^6	7	2^{-86}	202	7	2^{-88}	202
	10^9	12	2^{-85}	321	11	2^{-81}	286

Robustness bounds

Methodology:

- Set the testing time t
- Set the number of runs n
- Choose λ

$$\Pr[M(x) \neq D(x)] = \left(\frac{n}{t}\right)^{\lambda/2} \leq 2^{-80}$$

Changing the blockcipher allows to either:

- Reduce λ
- Get smaller bound

Robustness bounds

t [days]	**n** [bits]	**AES**			**Mysterion**		
		λ	**ROB.**	**Area** [GEs]	λ	**ROB.**	**Area** [GEs]
1	10^3	6	2^{-95}	181	5	2^{-81}	144
	10^6	8	2^{-86}	224	8	2^{-89}	224
	10^9	15	2^{-85}	380	14	2^{-84}	361
7	10^3	5	2^{-86}	144	5	2^{-88}	144
	10^6	7	2^{-86}	202	7	2^{-88}	202
	10^9	12	2^{-85}	321	11	2^{-81}	286

Outline

Introduction

Private Circuits 3

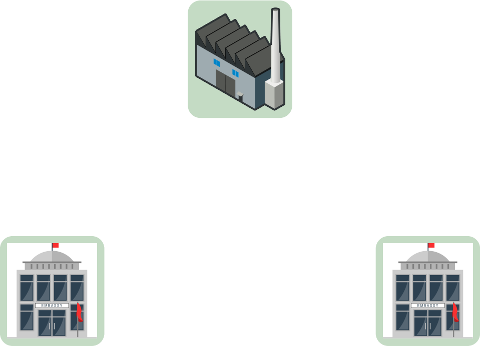
Targeted algorithm

Hardware Design

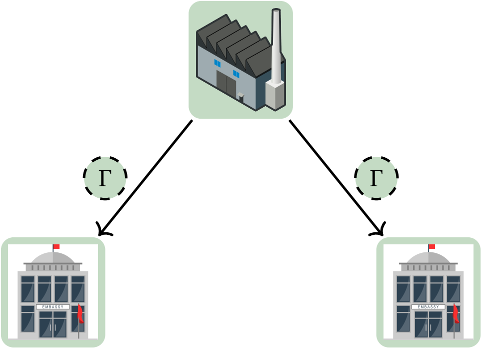
Conclusion

Conclusion

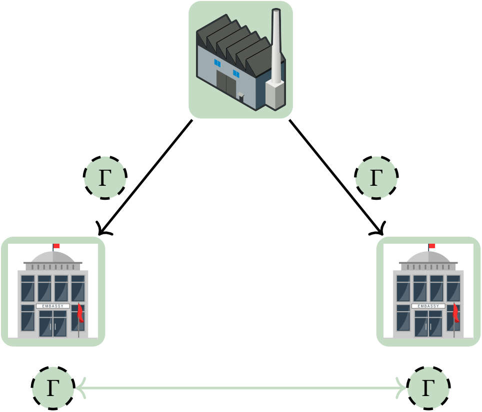
Conclusion



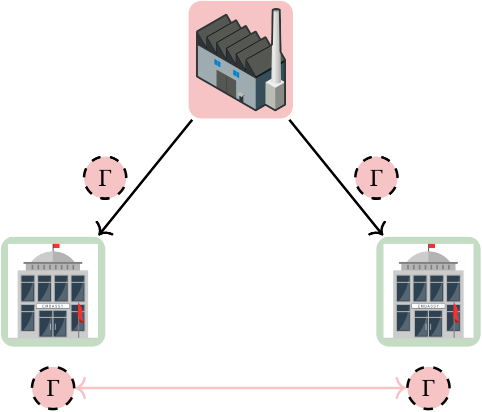
Conclusion



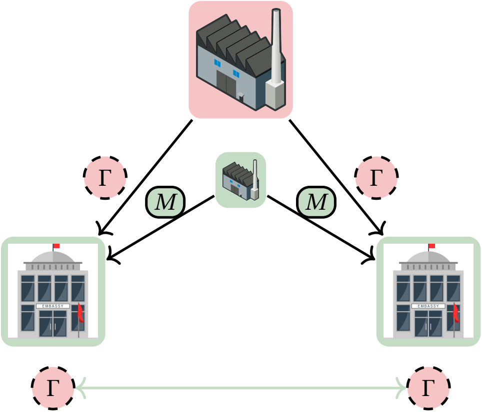
Conclusion



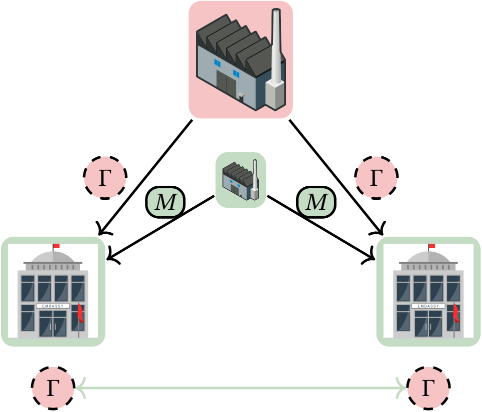
Conclusion



Conclusion



Conclusion



Thank you !

olivier.bronchain@uclouvain.be